Generic load balancing for HPF programs: Application to the Flame Simulation kernel

Gabriel Antoniu, Luc Bougé, Christian Pérez

Generic load balancing for HPF programs: Application to a flame simulation kernel

G. Antoniu L. Bougé C.Perez
LIP, ENS Lyon
46 Allée d'Italie, 69364 Lyon Cédex 07, France
{*Gabriel.Antoniu,Luc.Bouge,Christian.Perez*}*@ens-lyon.fr*

Abstract

This paper describes a scheme for implementing *generic* load balancing policies for irregular HPF applications. To this purpose, preemptive abstract processor migration appears to be a good support. Its implementation is based on fully preemptive thread migration. We report some encouraging results obtained with a modified version of Adaptor on a HPF flame simulation code, one of the motivating applications of HPF 2.0.

1 Introduction

HPF compilers have proven able to deal with regular computations and communications, whereas uneven load distributions usually limit the efficiency of HPF applications. A classical approach to irregularity relies on data distributions (Fig. 1). In HPF 1 [7] only the cyclic distribution is available. The approved extension of HPF 2 [8] proposes several irregular data distribution schemes issued from previous research efforts in irregular distribution [6]. An application is divided into phases and data may be redistributed at the beginning of each phase according to the most suitable (possibly irregular) pattern.

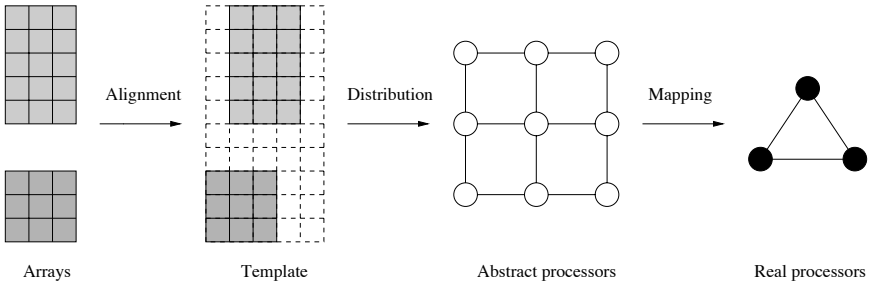

Figure 1: The HPF model

If these techniques may lead to accurate data distribution, there still remain some drawbacks. To use irregular data distributions, the HPF programmer needs to characterize in advance the load imbalance, to be able to choose the most appropriate (possibly irregular) distribution pattern. A shortcoming of the data redistribution technique is that the load balancing policy is pseudo-dynamic and that explicit calls to the data

flames. Each loop iteration contains two phases. The first phase is a stencil computation. The computations are balanced and neighborhood communications are needed. The second phase does not involve any communication, but the computation cost is irregular in space and in time. Figure 2 outlines the flame simulation code structure.

```
!lb$ begin work stealing with THRESHOLD, MAX, FREQUENCY
     do time = 1, timesteps
C       Convection phase
        x(2:NX-1,2:NY-1) = x(2:NX-1,2:NY-1) + F(z(2:NX-1,2:NY-1),
    &      y(1:NX-2,2:NY-1),y(3:NX,2:NY-1),y(2:NX-1,1:NY-2),y(2:NX-1,3:NY))
        y = x
C       Reaction phase
        forall(i=1:NX, j=1:NY) z(i,j) = Adaptative_Solver(x(i,j))
     end do
!lb$ end work stealing
```

Figure 2: The flame simulation kernel code.

The difficult point comes from the different requirements of the two phases. A regular distribution is best suited for the first phase whereas the second needs irregular distribution.

3.2 Data distributions issues

Orlando and Perego studied the flame simulation code in depth [12]. They simulate the code a compiler can generate using their SUPPLE runtime and hand-coded directly in MPI the communication code. Only limited classes of HPF loops are supported. Communications involving migrated tasks are not supported either. To communicate, tasks have to move back to their original node.

For the flame simulation code there is no initial data distribution able to ensure the efficient execution of the whole loop. The block distribution is well suited for regular data sets. Communications are minimal but computations are not balanced for irregular computations. The cyclic distribution succeeds in producing load balanced computation. However, this distribution has two drawbacks: communications are expensive because of the huge amount of data transfered; also a huge amount of memory is wasted for communication management. If the two dimensions are cyclicly distributed, only 20 % of the memory of a node is available for data. Redistributing the data for each phase allows efficient execution of both phases. The drawback is that there are two redistributions per iteration.

Orlando and Perego found out that dividing the work of each iteration in tasks and using a work stealing load balancing policy leads to the best performance. Initially, the data are block distributed. Communications are thus minimized and no overhead is introduced when the computations are regular. When computations become irregular, the underloaded nodes send messages to the other nodes to ask for work.

We have taken a similar approach, but with several original points. First, our load balancing module has been fully integrated into an HPF compiler. Also, the work stealing algorithm is implemented using preemptive abstract processor migration. Unlike the previous work, we are not limited to a particular kind of loop; our scheme is *generic*. Moreover, communication between abstract processors is supported irrespective of their location.

3.3 Experiments

The machine used for the experiments is a 12 node cluster of 200 MHz PentiumPro processors interconnected by a Myrinet network. We used the MPI-BIP communication library. It is an implementation of MPICH on top of BIP [13]. The network latency in this setup is 11 μs and the bandwidth is 66 MB/s. If BIP is used directly, the latency is 6 μs and the bandwidth is 125 MB/s.

Since the application behavior is dynamic, the load distribution may change from regular to highly irregular. We have considered three data sets, which differ in their irregularity degree. Also, the optimal time for each data set is different. So, execution times across each data set are hardly comparable.

Each data set has been run with different data distributions and load balancing policies. The first two lines of Table 1 display the results for static HPF distribution (block and cyclic). Then, a data redistribution has been used between the two phases. The first phase is executed with a block distribution while the second uses a cyclic distribution. So, two redistributions are performed per iteration. These three experiments have been done with the original Adaptor compiler. Finally, the last two lines of the table present results obtained with the modified version of Adaptor. The fourth line corresponds to block distribution and 128 abstract processors. The fifth is for the same configuration plus the work stealing policy.

Distribution Mode	Data Set Irregularity		
	None	Medium	High
Original ADAPTOR runtime			
Block	3.90	5.10	16.15
Cyclic	8.20	7.06	8.40
Redistribution	6.36	5.21	6.56
Modified ADAPTOR runtime, 128 abstract processors			
Block, no load balancing	3.93	3.78	9.21
Block, Work Stealing	4.30	4.16	8.61

Table 1: Time in seconds for different distributions on the flame simulation benchmark for 8 processors with various initial grids of size 1600 x 1600

These preliminary results confirm our expectations. The block distribution is not adequate when the computations are unbalanced. The cyclic distribution induces a high communication cost. Redistribution performs well. Using multithreading enables an overlap between computations and communications resulting in significantly improved performance for the the `BLOCK` distribution in irregular cases. Adding a generic work stealing strategy induces a slight overhead, but further improves performance in highly irregular cases.

4 Conclusion

We have shown that abstract processor migration is an interesting *generic* method to load balance HPF applications. To validate the usefulness of preemptive abstract processor migration, we have benchmarked a flame simulation kernel code, which is part of the motivating applications of HPF-2. The solution based on a work stealing strategy provides a

means of getting near-optimal performance *irrespective* of their degree of irregularity and with no modification of the source code.

We intend to test more applications using this approach. Also, we have built a multithreaded version of the Adaptor 6 HPF compiler, which will enable comparison tests using HPF 2 distribution patterns, such as `CYCLIC(N)` and `INDIRECT`.

References

[1] G. Antoniu, L. Bougé, and R. Namyst. An efficient and transparent thread migration scheme in the PM2 runtime system. In *Proc. 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP '99)*, volume 1586 of *Lect. Notes Comp. Science*, pages 496–510, San Juan, Puerto Rico, April 1999. IEEE TCPP and ACM SIGARCH, Springer-Verlag.

[2] G. Antoniu and C. Perez. Using preemptive thread migration to load-balance data-parallel applications. In *Euro-Par '99: Parallel Processing*, Lect. Notes Comp. Science, Toulouse, France, September 1999. Springer-Verlag. To appear.

[3] R. Blumofe and C. Leiserson. Scheduling multithreaded computations by work stealing. In *35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 356–368, Santa Fe, New Mexico, November 1994.

[4] L. Bougé, P. Hatcher, R. Namyst, and C. Perez. A multithreaded runtime environment with thread migration for a HPF data-parallel compiler. In *The 1998 Intl Conf. on Parallel Architectures and Compilation Techniques (PACT '98)*, pages 418–425, Paris, France, October 1998. IFIP WG 10.3 and IEEE.

[5] T. Brandes and F. Zimmermann. Adaptor - A transformation tool for HPF programs. In K. M. Decker and R. M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 91–96. Birkhäuser Verlag, April 1994.

[6] B. Chapman, H. Zima, and P.Mehotra. Extending HPF for advanced data-parallel applications. *IEEE Parallel and Distributed Technology*, 2(3):59–70, 1994.

[7] High Performance Fortran Forum. *High Performance Fortran Language Specification.* Rice University, Houston, Texas, November 1994. Version 1.1.

[8] High Performance Fortran Forum. *High Performance Fortran Language Specification.* Rice University, Houston, Texas, October 1996. Version 2.0.

[9] S. Goto, A. Kubota, T. Tanaka, M. Goshima, S. Mori, H. Nakashima, and S. Tomita. Optimized code generation for heteregeneous computing environment using parallelizing compiler TINPAR. In *Proc. 1998 Int. Conf. Parallel Architectures and Compilation Techniques (PACT'98)*, pages 426–433, ENST, Paris, France, October 1998. IFIP WG 10.3 and IEEE.

[10] HPF Forum. HPF-2 Scope of activities and motivating applications, November 1994. Ver. 0.8.

[11] R. Namyst and J.-F. Méhaut. PM2: Parallel multithreaded machine. A computing environment for distributed architectures. In *Parallel Computing (ParCo '95)*, pages 279–285. Elsevier Science Publishers, September 1995.

[12] S. Orlando and R. Perego. A comparison of implemantation strategies for nonuniform data-parallel computations. *J. Parallel Distrib. Comp.*, 52(2):132–149, 1998.

[13] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW '98)*, volume 1388 of *Lect. Notes Comp. Science*, pages 472–485. IEEE, Springer-Verlag, April 1998.